

## REMARKS

The March 13, 2002, Office Action (the "Office Action") in this application rejected Claims 1-3 and 5-17 under 35 U.S.C. § 103(a) as being unpatentable in view of the teachings of U.S. Patent No. 5,392,433 issued to Hammersley et al. (hereinafter "Hammersley et al."), taken in view of the statements made in portions of this application, more particularly, statements made at page 11, lines 22-25; page 12, lines 1-20; page 13; page 25; and page 14, lines 1-7. In addition, Claim 4 was rejected under 35 U.S.C. § 103(a) as being unpatentable in view of the teachings of Hammersley et al., taken in view of the teachings of U.S. Patent No. 6,119,173 issued to Pullen et al. (hereinafter "Pullen et al."). Furthermore, Claims 18 and 19 were rejected under 35 U.S.C. § 103(a) as being unpatentable in view of the teachings of Hammersley et al., taken in view of the statements referenced above, and U.S. Patent No. 6,122,631 issued to Berbec et al. (hereinafter "Berbec et al."). Portions of applicant's specification were objected to for lacking certain information, namely, patent application serial numbers. This amendment amends Claims 1, 5, 9, and 14 to clarify the subject matter of the claimed invention. In addition, minor amendments have been made to the specification directed to inserting the required patent application serial numbers for the patent applications that have been incorporated by reference.

Prior to discussing in detail why applicant believes that all of the claims in this application are allowable, a brief description of applicant's invention and a brief description of the teachings of the cited and applied references are provided. The following background and the discussions of the disclosed embodiments of applicant's invention and the teachings in the cited and applied references are not provided to define the scope or interpretation of any of the claims of this application. Instead, such discussions are provided to help the United States Patent and Trademark Office (hereinafter "the Office") better appreciate important claim distinctions discussed thereafter.

LAW OFFICES OF  
CHRISTENSEN O'CONNOR JOHNSON KINDNESS<sup>llc</sup>  
1420 Fifth Avenue, Suite 2800  
Seattle, Washington 98101  
206.682.8100

## Summary of Processes and Threads

Early operating systems allowed users to run only one program at a time. Users ran a program, waited for it to finish, and then ran another one. Modern operating systems allow users to execute (run) more than one program at a time or even multiple copies of the same program at the same time. This change highlights a subtle distinction: the difference between a program and a process. A program is a static sequence of instructions whereas a process is the dynamic invocation of a program along with the system resources required for the program to run. Thus, a process, in the simplest terms, is an executing program.

Processes include one or more threads. A thread is the basic unit used by the operating system to allocate processor time. A thread can include any part of the process code, including parts currently being executed by another thread. A processor is capable of executing only one thread at a time. However, a multitasking operating system, i.e., an operating system that allows users to run multiple programs, appears to execute multiple programs at the same time. In reality, a multitasking operating system continually alternates between programs, executing a thread from one program, then a thread from another program, etc. As each thread finishes its subtask, the processor is given another thread to execute. The extraordinary speed of the processor provides the illusion that all of the threads execute at the same time. Multitasking increases the amount of work a system accomplishes because most programs do not require that threads continuously execute. For example, periodically, a thread stops executing and waits while a slow resource completes a data transfer or while another thread is using a resource it needs. When one thread must wait, multitasking allows another thread to execute, thus taking advantage of processor cycles that would otherwise be wasted.

While the terms multitasking and multiprocessing are sometimes used interchangeably, they have different meanings. Multiprocessing requires multiple processors. If a machine has

only one processor, the operating system can multitask, but not multiprocess. If a machine has multiple processors, the operating system can both multitask and multiprocess.

In summary, an operating system having a process to run presumes the existence of a program from which the process is invoked because the program is the static aspect of the process and the process is the dynamic aspect of the program. Programs are typically created in an application development environment using an application development language. An application development environment is an integrated suite of applications for use by software developers to create programs. Typical components of application development environments include a compiler, a file browsing system, a debugger, and a text editor for use in creating programs. An application development language is a computer language designed for creating programs. The present invention as summarized below focuses on an application development environment, which in other words is a programming environment.

#### Summary of the Invention

Applicant's invention is directed to improving the development of programs so that programs may be developed in a more efficient and bug-free manner. As programs have become increasingly complex, programming environments are becoming a factor in the length of time it takes to create new programs and the number of bugs that occur as new programs are executed in conjunction with other programs. Traditionally, programs are developed in a linear manner. Unfortunately, a linear programming environment is often undesirable when highly complex computer programs that must run concurrently with other programs are to be developed. Another programming environment is the message-driven environment. In a message-driven programming environment, different objects interface with other objects via messages. Such messages typically are complex structures. To use such messages, the context must be unpacked from a message prior to the execution of an action. The message-driven programming environment as well as the linear programming environment are fragile and render development

of programs that must run concurrently with other programs potentially more difficult and bug-laden than is desired.

To solve or to reduce the foregoing problems, applicant's invention provides an asynchronous programming environment. More specifically, applicant's invention provides a dynamic object storage scheme for storing a plurality of objects, a dynamic dispatch scheme based on the plurality of objects, and an object recognition scheme to describe the plurality of objects. The dynamic object storage scheme stores the plurality of objects so that each object can be accessed as necessary by different threads within the asynchronous programming environment. The dynamic object storage scheme is dynamic in that objects can be created and removed as necessary during the execution of tasks within the asynchronous programming environment. The dynamic dispatch scheme is designed to invoke an action based on the plurality of objects stored by the dynamic object storage scheme. Each action that is invoked by the dynamic dispatch scheme falls into one of a plurality of categories. Actions may be classified differently at different times.

The plurality of categories include three categories—actions needing precisely one object, actions needing more than one object, or actions not needing an object at all. Actions needing precisely one object include message handlers, and other actions that do not need any object for their execution are generally used to create objects, such as default constructors and real-time routines. Actions that require multiple objects for their dispatch generally combine objects and perform tasks as designed by the programmer.

Other aspects of present invention include an object recognition scheme that describes the plurality of objects stored by the dynamic object storage scheme. The description of objects allows the asynchronous programming environment to determine whether a described object fits the needs of an application programming interface. In other words, when an application programming interface is presented to the asynchronous programming environment, it is not

always clear which objects in a depository of objects fit the presented application programming interface. The object recognition scheme of the present invention allows one or more objects to be identified, thereby determining the suitability of one or more objects to the requirements of the application programming interface.

Unlike prior programming environments, an asynchronous programming environment allows thread-agnostic programs to be developed. Such programs have symmetric multithreading capabilities, i.e., they can be scaled to use the most of available processor processing power. Symmetric multithreading is expected to reduce instruction cache misses on hardware implementations that handle multiple instruction streams at once. An asynchronous programming environment is more efficient than a message-driven programming environment because execution is driven via presence of objects so that little or no translation is needed to recover context of execution. These attributes of an asynchronous programming environment allow programs to be developed in a more efficient and more bug-free manner.

The technical differences between an asynchronous programming environment and an asynchronous program (discussed below) need to be understood in order to understand the present invention.

#### Summary of Hammersley et al.

Unlike applicant's invention, Hammersley et al. is directed to an operating system that allows intraprocess locking of a shared resource in a computer system. The operating system of Hammersley et al. purportedly supports multiple processes. Each process is capable of being executed by a plurality of asynchronous programs. In addition to ensuring that only one process may use a shared resource, the operating system of Hammersley et al. purportedly grants access to a shared resource at the level of a program executing within a process. When an asynchronous program accesses a shared resource, the program requests that the operating system lock out all other programs executing asynchronously within that process, as well as other processes

executing within the operating system, from that shared resource. Hammersley et al. is not at all concerned with providing a programming environment in which programs are created.

There is a technical difference between an "asynchronous programming environment" (the present invention) and an "asynchronously executing programs" (Hammersley et al.) For example, the present invention includes a dynamic dispatch scheme. Hammersley et al. does not discuss or even allude to a dynamic dispatch scheme. In this regard, as discussed above, a dynamic dispatch scheme can invoke an action that may fall into one of a plurality of categories. The categories include needing one object, needing more than one object, or not needing an object at all. The Office cites Figures 2B, 3B, and 3C of Hammersley et al. as disclosing a dynamic dispatch scheme. Applicant submits that the specification of Hammersley et al. does not support this proposition. The Brief Description of the Drawings of Hammersley et al., describes Figure 2B as showing an example of a process level locking of a shared computer resource in a prior art computer environment that is capable of supporting multiple processes. Figure 3B of Hammersley et al. is described as showing an example of the failure of process level locking of a shared computer resource in a hypothetical computer environment that is capable of supporting multiple asynchronous programs running within a single process. And Figure 3C of Hammersley et al. is described as showing an intraprocess or program level locking of a shared computer resource in the computer environment that is capable of supporting multiple asynchronous programs running within a single process. Besides the fact that none of the cited figures even remotely suggests dynamic dispatch scheme of the type contemplated by the present invention, each asynchronous program executed by the operating system of Hammersley et al. knows precisely the shared resource which the program needs to access, thereby having no need for a dynamic dispatch scheme.

The operating system of Hammersley et al. also lacks an object recognition scheme. The object recognition scheme of the present invention identifies one or more objects and determines

whether these objects fit a given application programming interface. The Office cited portions (lock table 80 and column 4, lines 44-46) of the specification of Hammersley et al. as disclosing an object recognition scheme. Applicant respectfully submits that this conclusion is incorrect. The lock table of Hammersley et al. contains a plurality of hash entries. Each hash entry contains a resource address field and a pointer field. The resource address field contains the address of the resource for which locking is desirable, while the pointer field points to a lock entry vector for a resource address. The lock table lacks any description of an object sufficient to allow an asynchronous programming environment to determine whether a given object fits a given application programming interface. Therefore, Hammersley et al. does not teach an object recognition scheme of the type contemplated by the present invention.

The Office noted that Hammersley et al. fails to disclose a dynamic object storage scheme for storing a plurality of objects. The Office overcomes this failure by noting that applicant has stated that a dynamic object storage scheme is known within the art. While the Office is correct in concluding that, broadly, a dynamic object storage scheme is known within the art, a *prima facie* case of obviousness cannot be established by using applicant's disclosure as a motivation to combine a dynamic object storage scheme with other inventive subject matter taught by applicant. See M.P.E.P. § 2143 (the teaching or suggestion to make the claimed combination and the reasonable expectation of success must both be found in the prior art, not in applicant's disclosure. *In re Vaeck*, 947 F.2d 488, 20 U.S.P.Q.2d 1438 (Fed. Cir. 1991)). Thus, not only has the Office failed to show that Hammersley et al. teaches or even remotely suggests the dynamic dispatch scheme and the object recognition scheme of the present invention, the Office has failed even to provide a motivation for the proposed combination independent from applicant's disclosure. Therefore, no *prima facie* case of obviousness has been established.

### Summary of Pullen et al.

Pullen et al. discloses a system for managing a plurality of applications and communication in a distributed telecommunication switch. The distributed telecommunication switch includes a service unit and at least one delivery unit. The system includes a services element residing in the service unit operable to provide a plurality of services to the plurality of applications, and an application procedure interface residing in the service unit operable to serve as an interface between the plurality of applications and the services element. The services element provides the plurality of services in accordance with messages received in a message distribution scheme. For example, a control message may be generated to notify certain events, including stay transition, shut down, restart, and logging level change. In other words, events are packed into messages and forwarded to the services element for the services element to unpack before the services element provides the requested services.

In contrast, the present application explains (see e.g., page 2, lines 20-24 of the pending application) that the use of messages, which are complex structures, for the purpose of unpacking or reconstituting the context prior to the execution of an action is problematic. In other words, Pullen et al. precisely teaches opposite and away from the present invention. Thus, there is no benefit to combine Pullen et al. and Hammersley et al. if the combination were possible, which applicant specifically denies. As indicated above, there is a defect in the combination of Hammersley et al. and applicant's disclosure. Pullen et al. does not cure this defect. Thus, even if Hammersley et al. and Pullen et al. were combinable, which applicant specifically denies, the resulting combination would not anticipate applicant's invention.

### Summary of Berbec et al.

The system of Berbec et al. provides dynamically controlling access to files in a client/server system. A client wanting access to a file first requests a token from an object server. The object server generates the token as the function of the file name and provides the



token to both the client and the file system. Upon receiving the token from the object server, the client presents an access request to the file system, using the token rather than the file name as an identifier. The file system compares the received identifier with the file names in the specified directory as well as with a list of valid tokens that it maintains. If the identifier corresponds to a valid token for a file in the directory, the access request is granted. Otherwise, the access request is denied.

Berbec et al. fails to disclose a dynamic object storage scheme for storing a plurality of objects, a dynamic dispatch scheme based on a plurality of objects, and an object recognition scheme to describe the plurality of objects. As explained above, the combination of Hammersley et al. and applicant's disclosure is insufficient to anticipate applicant's invention. Berbec et al. does not cure this defective combination. Thus, even if the combination of Hammersley et al., applicant's disclosure, and Berbec et al. were possible, which applicant specifically denies, the resulting combination would not anticipate applicant's invention.

#### The Claims Distinguished

The Office has failed to show, and applicant is unable to find, where any of the cited and applied references, either alone or in combination, disclose the subject matter of the claimed invention. Among other differences, none of the applied and cited references teaches "a dynamic dispatch scheme being capable of invoking an action that belongs to one of a plurality of categories, the plurality of categories including needing one object, needing more one than one object, and needing no object" as recited in amended independent Claim 1. Similar language is now included in amended independent Claims 5, 9 and 14. Because the system of Hammersley et al. is not employed in an asynchronous programming environment, Hammersley et al. does not recognize the problems of developing thread-agnostic programs and the need to eliminate or reduce recovery of context of execution. Because each program in a process described by

Hammersley et al. knows precisely the resource in which a program needs access, there is no reason for Hammersley et al. to provide a dynamic dispatch scheme.

Applicant is also unable to find where any of the cited and applied references discuss or even remotely suggest "object recognition scheme for providing a description of each object of the plurality of objects, the description to allow a determination of whether an object being described by the description fits an application programming interface" as recited in amended Claim 1. Similar language is now included in amended independent Claims 5, 9 and 14. As explained above, the system of Hammersley et al. lacks an object recognition scheme. Hammersley et al. focuses on intralocking of resources between programs within a single process. The focus of the present application is completely different. There is no need for Hammersley et al. to provide an object recognition scheme because Hammersley et al. does not require one. Berbec et al. teaches the use of tokens rather than file names as identifiers, but lacks any teaching regarding the use of a description of an object to determine whether the object fits an application programming interface.

Because the Office has failed to state a *prima facie* case of obviousness, the rejection should be withdrawn. Amended independent Claims 1, 5, 9, and 14 are clearly patentably distinguishable over the cited and applied references. Claims 2-4, 6-8, 10-13, and 15-19 are allowable because they depend from allowable independent claims and because of the additional limitations added by those claims. Consequently, reconsideration and allowance of Claims 1-19 is respectfully requested.

#### Allegedly Admitted Prior Art

The Office has alleged that various portions of applicant's disclosure are admitted prior art. Applicant respectfully traverses such an allegation. For example, the Office has indicated that applicant's disclosure teaches that a recyclable locking mechanism is prior art. Applicant respectfully disagrees. No reasonable interpretation of the language of applicant's disclosure

LAW OFFICES OF  
CHRISTENSEN O'CONNOR JOHNSON KINDNESS<sup>PC</sup>  
1420 Fifth Avenue, Suite 2800  
Seattle, Washington 98101  
206.682.8100

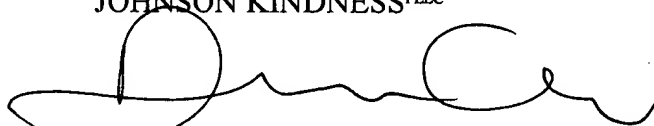
provides support for the conclusion that a recyclable locking mechanism is prior art. The recyclable locking mechanism referenced in the present application is the subject of another copending patent application, which is not prior art. While the present application indicates that a dynamic object storage scheme is known within the art and that threads are known within the art, there is no express or implied admission that a recyclable locking mechanism is so known. Accordingly, applicant respectfully requests that the Office withdraw these allegations.

#### CONCLUSION

In view of the foregoing remarks, applicant submits that all of the claims in the present application, as amended, are clearly patentably distinguishable over the teachings of Hammersley et al., Pullen et al., and Berbec et al. taken alone or in combination, including a combination with applicant's alleged admissions. Thus, applicant submits that this application is in condition for allowance. Reconsideration and reexamination of the application, allowance of the claims, and passing of the application to issue at an early date are solicited. If the Examiner has any remaining questions concerning this application, the Examiner is invited to contact the applicant's undersigned attorney at the number below.

Respectfully submitted,

CHRISTENSEN O'CONNOR  
JOHNSON KINDNESS<sup>PLLC</sup>



D.C. Peter Chu  
Registration No. 41,676  
Direct Dial No. 206.695.1636

#### **CERTIFICATE OF MAILING**

I hereby certify that this correspondence is being deposited with the U.S. Postal Service in a sealed envelope as first class mail with postage thereon fully prepaid addressed to Box Non-Fee Amendment, U.S. Patent and Trademark Office, P.O. Box 2327, Arlington, VA 22202, on the date specified below.

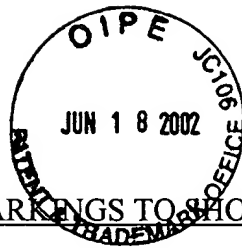
Date:

June 13, 2002

Lindy A. Norton

PCC:clm

LAW OFFICES OF  
CHRISTENSEN O'CONNOR JOHNSON KINDNESS<sup>PLLC</sup>  
1420 Fifth Avenue, Suite 2800  
Seattle, Washington 98101  
206.682.8100



VERSION WITH MARKINGS TO SHOW CHANGES MADE JUNE 13, 2002

In the Specification:

The section beginning at page 1, line 3, has been amended as follows:

[OTHER APPLICATION] CROSS-REFERENCES TO RELATED APPLICATIONS

The present application incorporates by reference the coassigned and copending U.S. [applications] Application No. 09/217,389, filed December 21, 1998, entitled "A Recyclable Locking for Multi-Threaded Computing Environments," [≅ [docket no. 777.154US1],] and U.S. Application No. 09/217,402, filed December 21, 1998, entitled [and] "A Token-Based Object Description," [≅ [docket no. 777.155US1].]

The paragraph beginning at page 12, line 4, has been amended as follows:

The dynamic object storage scheme 200 stores the plurality of objects 200 so that they may be accessed as necessary by different threads within the asynchronous programming environment. The object storage scheme is dynamic in that objects may be created and removed as necessary during the execution of tasks within the asynchronous programming environment. Such schemes are known within the art. In one embodiment, the plurality of objects 200 can be locked in accordance with[,] a recyclable locking mechanism 204, as is described in the copending and coassigned [application] U.S. Application No. 09/217,389, filed December 21, 1998, entitled "A Recyclable Locking for Multi-Threaded Computing Environments," [≅ [docket no. 777.154US1],] which has already been incorporated by reference. The mechanism uses the plurality of lock objects 206 to accomplish the locking of the objects 200 for exclusive or non-exclusive access by threads, as that term is known within the art.

LAW OFFICES OF  
CHRISTENSEN O'CONNOR JOHNSON KINDNESS<sup>LLC</sup>  
1420 Fifth Avenue, Suite 2800  
Seattle, Washington 98101  
206.682.8100

The paragraph beginning at page 13, line 24, has been amended as follows:

Finally, the object recognition scheme 212 provides for the description of the plurality of objects. The description of objects is necessary so that functions such as application, programming interfaces [(API=s)] (APIs) are able to determine whether a given object fits a given API. Such descriptions may include, for example, a hierarchical tree structure of object types, as known within the art, which require traversal to determine the description of a given object. However, in one embodiment, the scheme 212 utilizes a plurality of tokens 214, such that each object within the plurality [-] of objects 200 is describable with a sequence of tokens, where each token relates to an attribute of the object, such as object type. Such token-based description is pursuant to description provided in the copending and coassigned [application] U.S. Application No. 09/217,402, filed December 21, 1998, entitled "A Token-Based Object Description," [≡ [docket no. 777.155US1],] which has already been incorporated by reference.

The paragraph beginning at page 15, line 22, has been amended as follows:

Referring now to FIG. 3, a flowchart of a method according to one embodiment of the invention is shown. In 300, a plurality of objects is stored via a dynamic object storage scheme, for example, as has been described in the previous section of the detailed description. In one embodiment, such storage is accomplished such that the objects are accessed utilizing a recyclable locking mechanism as described in the copending and coassigned [application] U.S. Application No. 09/217,389, filed December 21, 1998, entitled "A Recyclable Locking for Multi-Threaded Computing Environments," [≡ [docket no. 777.154US1],] which has previously been incorporated by reference.

The paragraph beginning at page 16, line 17, has been amended as follows:

In 304, finally, each of the plurality of objects is described utilizing an object recognition scheme, as has been described in the previous section of the detailed description. Such description provides for the testing of the objects against functions such as application programming interfaces [(API=s)] (APIs), so that it can be determined efficiently whether a given object can be executed against a given API. In one embodiment, the object recognition scheme includes describing each of the objects as a series of tokens, where each token relates to an attribute of the object (for example, the type of the object), pursuant to the copending and coassigned [application] U.S. Application No. 09/217,402, filed December 21, 1998, entitled "A Token-Based Object Description," [ $\cong$  (docket no. 777.155US1),] which has already been incorporated herein by reference.

In the Claims:

1. (Amended) An asynchronous programming environment, comprising:  
a dynamic object storage scheme for storing a plurality of objects;  
a dynamic dispatch scheme [based on the plurality of objects] for invoking an action that belongs to one of a plurality of categories, the plurality of categories including needing one object, needing more than one object, and needing no object; and[,]  
an object recognition scheme [to describe the plurality of objects] for providing a description of each object of the plurality of objects, the description allowing a determination of whether an object described by the description fits an application programming interface.

5. (Amended) A method comprising:  
storing a plurality of objects via a dynamic object storage scheme;

dispatching at least one of the plurality of objects via a dynamic dispatch scheme based on events from at least one of the plurality of objects, the dynamic dispatch scheme capable of invoking an action that belongs to one of a plurality of categories, the plurality of categories including needing one object, needing more than one object, and needing no object; and[,]

describing each of the plurality of objects utilizing an object recognition scheme, the object recognition scheme providing a description of each object of the plurality of objects, the description allowing a determination of whether an object described by the description fits an application programming interface.

9. (Amended) A computer comprising:

a processor;

a computer-readable medium; and[,]

an asynchronous programming environment executed by the processor from the medium, the environment comprising:

a dynamic object storage scheme for storing a plurality of objects;

a dynamic dispatch scheme based on events from at least one of the plurality of objects, for invoking an action that belongs to one of a plurality of categories, the plurality of categories including needing one object, needing more than one object, and needing no object; and[,]

an object recognition scheme [to describe the plurality of objects,]for providing a description of each object of the plurality of objects, the description allowing a determination of whether an object described by the description fits an application programming interface.

14. (Amended) A computer-readable medium having a computer program stored thereon for execution on a computer, the computer program providing an asynchronous programming environment comprising:

a dynamic object storage scheme for storing a plurality of objects;

a dynamic dispatch scheme based on events from at least one of the plurality of objects for invoking an action that belongs to one of a plurality of categories, the plurality of categories including needing one object, needing more than one object, and needing no object; and[,]

an object recognition scheme [to describe the plurality of objects] for providing a description of each object of the plurality of objects, the description allowing a determination of whether an object described by the description fits an application programming interface.